[1] TITLE OF THE INVENTION:

[2] TABLE LOOKUP MECHANISM FOR ADDRESS RESOLUTION

[3] RELATED APPLICATIONS:

[4] This application claims the benefit of U.S. Provisional Patent Application No. 60/166,225 filed on November 18, 1999. The contents of this Provisional Patent Application is hereby incorporated be reference.

[5] BACKGROUND OF THE INVENTION

[6] FIELD OF THE INVENTION:

[7] The present invention is directed to a method and apparatus for a table lookup index that provides access to a table, such as an addressing table, in a fast and efficient manner. More specifically the table lookup index is for the transmission of data packets in a network switch.

[8] DESCRIPTION OF THE PRIOR ART:

[9] In the world of data transmission it is necessary to transmit data as quickly as possible, in order to have almost instantaneous access to information.

[10] Table lookups are frequently used to index information to give quick access to much needed data. For example in network switching it is imperative that when data is received in a port the appropriate port for output be identified in order to transfer data. Therefore table lookups are utilized to store information regarding incoming data that directs the incoming data to an appropriate port for output.

[11] In the field of table look-ups there are several methods used to access information. One method is a linear index.

[12] FIG. 1B illustrates a 64K table 100 that is linearly indexed. Index 105 for a 64K table is required to be 16 bits long (2^{16}=64K) to linearly access entries in the table. Since the index linearly accesses entries in the table, there is a one to one correspondence between each index and each table entry (i.e. each index has a corresponding table entry).

[13] FIG. 1C is an illustration of a linear index into a 64K table. Indexes I(1), I(2), I(3) ... are each linearly indexed into the 64K table giving a one to one correspondence between each index and each table entry. If indexes I(1), I(2), I(3) ... each are 16 bits long an entry can be found in 16 clock cycles using standard binary searching. However, when a new address must be learned by inserting an entry or deleting an entry, it will take a long time to delete or insert the address from or to the table because the index must be sorted after each insertion and each deletion.

In the worst case scenario each of the sixteen bit address indexes will have to be moved and sorted which will be time consuming. This sorting greatly degrades the performance of the switch. When the new address is inserted into the table, addresses will have to be moved up and/or down in order to make room for the new address. Therefore as the table size increases, the performance of the switch degrades.

[14] Other methods make use of pointers, hash functions or tree functions to remedy the problem. Most of these methods, when implemented, have a "bad" worst case performance - particularly for insert and delete operations. As noted above, as the size of the table increases, performance becomes degraded. Some of the main reasons for the poor results are that the table size directly affects performance and the methods require extra storage and data structures.

[15] SUMMARY OF THE INVENTION:

[16] The present invention is designed to overcome the deficiencies of the prior art and to increase the speed of data transmission.

[17] In one embodiment, the present invention is a method of performing a table look-up. The method has the steps of receiving data through an input source; parsing the data into an index portion and a corresponding bucket portion; indexing the index portion to the

corresponding bucket portion; and accessing table information stored in a look-up table using the bucket portion.

[18] In another embodiment, the present invention is a table look-up indexing device. The table look-up indexing device has a receiver that receives incoming data; a data parser that parses the data into an index portion and a corresponding bucket portion; an indexer that indexes the index portion to the bucket portion; and a lookup device that accesses a look-up table using the corresponding bucket portion.

[19] In a further embodiment, the present invention is a network switch having multiple ports used for receiving and exporting data. Each of the multiple ports are connected to one another through a communications medium. Multiple Address Resolution Logic (ARL) devices are connected to one of the multiple ports so that each of the multiple ports has a corresponding ARL device. Each of said multiple ARL devices is made up of a parser that parses data into an index portion and a corresponding bucket portion; an indexer that indexes said index portion to a corresponding bucket portion; and a look-up device that accesses table entries in a look-up table using the bucket portion.

[20] BRIEF DESCRIPTION OF THE DRAWINGS:

[21] The objects and features of the invention will be more readily understood with reference to the following description and the attached drawings, wherein:

[22] FIG. 1A is an illustration of a network switch having look-up tables (L Tables) and Address Resolution Logic (ARL);

[23] FIG. 1B is an illustration of a 16 bit index and a 64K table of the prior art;

[24] FIG. 1C is an illustration of a linear index into a 64K table of the prior art;

[25] FIG. 2A is an illustration of a 48 bit key being parsed into and index portion and a bucket portion to select a particular entry in a 64K table;

[26] FIG. 2B is an illustration of indexing a 64K table using an index portion and a bucket portion for multiple indexes;

[27] FIG. 3 is an illustration of a 48 bit key having a bucket size of 32 and being indexed into a 64K table;

[28] FIG. 4A is an illustration of the method steps of an embodiment of the present invention.

[29] FIG. 4B is an illustration of the method steps of another embodiment of the present invention.

[30] FIG. 5A is an illustration of a circuit receiving an address and using an index portion and bucket portion to identify a particular entry in a table to determine the appropriate output port for data transmission.

[31] FIG. 5B is an illustration of a network switch receiving an address and using an index portion and bucket portion to identify a particular entry in a table to determine the appropriate output port for data transmission.

[32] FIG. 6 is an illustration of a network switch in accordance with the present invention.

[33] DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS:

[34] The present invention is a method and apparatus that performs a table look-up that parses incoming data into an index portion and a bucket portion and uses the index portion and bucket portion to do a look-up.

[35] FIG. 1A illustrates a network switch 50 having 5 ports, 52, 54, 56, 58 and 60. Each of the 5 ports have corresponding switching tables 62, 64, 66, 68 and 70, respectively. Each of the switching tables 62, 64, 66, 68 and 70 has Address Resolution Logic (ARL) and Lookup Tables (L Tables). Each of the 5 ports are connected to one another through a communications line 72.

[36] When data is received by one of the ports 52, 54, 56, 58 or 60 the data is indexed by the corresponding ARL. Simultaneously the data is

sent to all the ports to determine which port the data should be sent to for output. When the output port is determined the ARL stores the output port data in the L Table using the index. In this way the output port for a particular type of data is "learned".

[37] Once the output port for a particular type of data is "learned", when a port receives data, the ARL indexes the data and "looks up" the proper output port in the look-up table, L Table. The data is then sent to the appropriate port for output and the step of sending data to every port is eliminated thereby speeding up the switching process.

[38] For Example, suppose initial data is received by port 52 and is to be output through port 60. Initially data is sent to all the ports to determine the proper output port. When the proper output port is determined, the initial data is sent to switching table 62 to be indexed by the ARL and the output port data is stored in the look-up table, L Table using the index created by the ARL. In the present case the L Table contains output port data directing any data with a particular index to be sent to port 60.

[39] When port 52 receives subsequent data, switching table 62 indexes the data and "looks up" the index in the look-up table, L Table. If the subsequent data has the same index as the initial data and is found in the look-up table, the look-up table will show that the subsequent data should

be sent to port 60 for output and the subsequent data will be automatically sent to port 60 for output.

[40] If the subsequent data is indexed and the index is not found in the look-up table, the subsequent data will be sent to all ports to determine the appropriate output port. The subsequent data will then be indexed and the output port information stored in a look-up table by switching table 62. The subsequent data is now "learned" and the next time incoming data comes in with the same index as the subsequent data, the incoming data will be immediately sent to the proper port for output since the data or index is now "learned".

[41] FIG. 2A illustrates a data structure 200 made up of 48 bits used as data for being received by ports 52, 54, 56, 58 and 60. This 48 bit data structure 200, commonly referred to as a key, is typically some type of address that is parsed into multiple groups. In the present case the data structure depicted in FIG. 2A is parsed into 3 segments; segment M, index segment I and bucket segment N. Bucket segment N is designated as the bucket, index segment I is designated as the index, and segment M is the remaining bits of the key.

[42] For a key size of 48 and a table size of 64K, 16 bits are used for indexing. The 48 bit key is parsed into 3 segments, segment M, index segment I and bucket segment N.

[43] FIG. 2B is an illustration of an index segment I(1) linearly indexed to a bucket segment N(1), an index segment I(2) linearly indexed to a bucket segment N(2) and an index segment I(3) linearly indexed to a bucket segment N(3)... Each index segment I selects a bucket segment N and the combination of index segment I and index segment N selects an entry in the table. Each bucket segment N can be programmed to have a particular size. Since the size of index segment I is dependent upon the size of bucket segment N, the size of index segment I will change accordingly.

[44] FIG. 3 illustrates a 48 bit key having a bucket size of 32 (i.e. bucket segment N is 5 bits long and 2^5=32) for a 64K table. Since it takes 16 bits (64K=2^{16}) to index the 64K table and the bucket size is 32 (5 bits long), index segment I is 11 bits long (16 bits - 5 bits = 11 bits) and segment M is 32 bits long (48 bits - 16 bits = 32 bits).

[45] Index segment I is called the index and is linearly indexed to a bucket which is designated as a bucket segment N. The combination of the index and bucket directly selects an entry in the table. Therefore once the index is selected it will take a maximum of five clock cycles to search the bucket and only 5 bits will have to be binary sorted for an insert or delete operation. Thus an insert or delete function is dependent on the bucket size which can be programmed to fit a particular need.

[46] It is noted that in some instances it is important to select bits from a key for index segment I that will have recurring index segments I for all incoming keys. This will allow for bucket sizes of 2 or greater which is more desirable for high speed switching.

[47] In some applications manufacturers of chips use a manufacturers identification number as part of the key. Therefore it may be valuable to use bits in the key used for the manufacturers identification number as part of the index segment I. Another useful method for indexing index segment I is to use an XOR index where the index is XORed with the remaining bits of the key.

[48] FIG. 4A is an illustration of the steps needed to index a look-up table in accordance with the present invention. In step 410 data is received as input. The data is then parsed, in step 420, into an index portion and a corresponding bucket portion. In step 430 the index portion is indexed to the bucket portion. In step 440 table information stored in a look-up table is accessed using the bucket portion.

[49] FIG. 4B has very similar steps to the steps illustrated in FIG. 4A. In step 450 data is received in a port. The data is then parsed, in step 460, into an index portion and a corresponding bucket portion. In step 470 the index portion is indexed to the bucket portion. The major difference from the steps illustrated in FIG. 4A and those illustrated in FIG. 4B is the

following final step. In step 480 table information stored in a look-up table is accessed using the bucket portion. However, if there is no table information stored in the look-up table for the bucket portion, the present invention stores table information into the look-up table corresponding to the bucket portion as depicted in step 490.

[50] In one embodiment of the invention a MAC address is received in a port of a network switch having multiple ports. The MAC address is parsed into an index portion and a bucket portion. The index portion is indexed to a bucket portion and the bucket portion is used to access a table entry in a look-up table to determine which port the MAC address and associated data should be sent to for output. If there is a table entry for the corresponding bucket portion the MAC address and associated data will be sent to the proper port for output. In the case where there is no table entry the port for output has to be "learned". This is accomplished by sending the MAC address to all ports in the switch. The proper port for output will send a message back that it is the proper port for output and this information is stored in the look-up table with the corresponding bucket. Therefore any subsequent MAC addresses having the same index and bucket will be able to access the table entry in the look-up table to determine the proper port for output without sending the MAC address to all the ports in the switch.

[51] FIG. 5A is an illustration of a switch 500. In order to transmit data at a very quick and efficient manner switch 500 learns which output port each incoming MAC addresses 505 is to be transmitted and stores this information in table 510. When a new MAC address 505 is received by switch 500, the MAC address is looked-up in table 510 and switched to the proper destination output port 540 or 545.

[52] An address 505 is received by switch 500 through an input port 515. the input port 515 transmits address 505 to parser 520. Parser 520 segments address 505 into three parts, segment 525 labeled as M, segment 530 labeled as I and segment 535 labeled as N. Segment I is linearly indexed to a bucket 535 and bucket 535 directly indexes a entry in table 510. In effect bucket 535 does a look up in table 510 in order to determine which of output ports 540 or 545 are to be used to transmit incoming data. The Index I merely selects a bucket.

[53] FIG. 6 is an illustration of a network switch 600 having a port 605, a port 610 and a port 615. Each of these ports are connected to one another through a communications line 620.

[54] Port 605 has a corresponding Address Resolution Logic device (ARL) 625 and a corresponding look-up table 630. ARL 625 is made up of a parser 635, an indexer 640 and a look-up device 645.

[55] Port 610 has a corresponding Address Resolution Logic device (ARL) 650 and a corresponding look-up table 655. ARL 650 is made up of a parser 660, an indexer 665 and a look-up device 670.

[56] Port 615 has a corresponding Address Resolution Logic device (ARL) 675 and a corresponding look-up table 680. ARL 675 is made up of a parser 685, an indexer 690 and a look-up device 695.

[57] In the event that port 605 receives a MAC address to be sent to port 610, the operation of the switch is as follows. A MAC address is received in port 605 of network switch 600 having multiple ports 605, 610 and 615. The MAC address is sent to ARL 625 to be processed. Parser 635 parses the MAC address into an index portion and a bucket portion. Indexer 640 indexes the index portion to a bucket in look-up 645. The bucket portion in look-up 645 is used to access a table entry in look-up table 630 to determine which port the MAC address and associated data should be sent to for output. If there is a table entry for the corresponding bucket portion the MAC address and associated data will be sent to the proper port for output. In this case the look-up table entry will direct the MAC address and associated data to output port 610. However, if there is no table entry the port for output has to be "learned". This is accomplished by sending the MAC address to all ports, ports 605, 610 and 615, in switch 600. The proper port for output will send a message

back that it is the proper port for output and this information is stored in look-up table 630 with the corresponding bucket. In this case port 610 will send port 605 a signal indicating that port 610 is the proper port for output and this information is stored in look-up table 630. Therefore any subsequent MAC addresses having the same index and bucket will be able to access the table entry in the look-up table to determine the proper port for output without sending the MAC address to all the ports in the switch.

[58] An advantage of the present invention is that insert and delete operations will no longer be dependent upon table size. As discussed above, address look-ups in the prior art are dependent upon table size. Therefore if the table size is 64K the index will be 16 bits long. Thus an insert or delete operation will involve sorting and moving 16 bit indexes.

[59] However, the present invention for a 64K table with a bucket size of 32 (5 bits) will only have an 11 bit index (16 bits - 5 bits = 11 bits). Therefore once a bucket is selected by an index, insert or delete operations will depend on the bucket size and involve the sorting and moving of, in this case, 5 bit bucket indexes which is much more efficient than the sorting and moving of 16 bit indexes.

[60] Although the invention has been described based upon the embodiments discussed above, it would be apparent to those skilled in the

art that certain modifications, variations and alternative constructions would be apparent, while remaining within the spirit and scope of the invention. For example instead of using a linear index an XOR index is appropriate in certain circumstances. Likewise the bucket and index sizes are programmable to accommodate different circumstances and be tailored to specific needs.